Heap

package main

import "fmt"

type MinHeap struct {

arr []int

}

func (h \*MinHeap) Insert(value int) {

h.arr = append(h.arr, value)

currentIndex := len(h.arr)-1

h.ShiftUp(currentIndex)

}

func (h \*MinHeap) ShiftUp(currentIndex int) {

parentIndex := h.Parent(currentIndex)

for currentIndex > 0 && h.arr[currentIndex] < h.arr[parentIndex] {

h.arr[currentIndex],h.arr[parentIndex] = h.arr[parentIndex],h.arr[currentIndex]

currentIndex = parentIndex

parentIndex = h.Parent(currentIndex)

}

}

func (h \*MinHeap) Remove(){

h.arr[0],h.arr[len(h.arr)-1] = h.arr[len(h.arr)-1], h.arr[0]

h.arr = h.arr[:len(h.arr)-1]

h.ShiftDown(0)

}

func (h \*MinHeap) ShiftDown(currIdx int) {

endIdx := len(h.arr)-1

leftIdx := h.LeftChild(currIdx)

for leftIdx <= endIdx {

rightIdx := h.RightChild(currIdx)

idxToShift := leftIdx

if rightIdx <= endIdx && h.arr[rightIdx] < h.arr[leftIdx]{

idxToShift = rightIdx

}

if h.arr[currIdx] > h.arr[idxToShift] {

h.arr[currIdx],h.arr[idxToShift] = h.arr[idxToShift],h.arr[currIdx]

currIdx = idxToShift

leftIdx = h.LeftChild(currIdx)

}else{

return

}

}

}

func (h \*MinHeap) Heapify(arr []int) {

for \_,n := range arr {

h.Insert(n)

}

}

func (h \*MinHeap) Display() {

for \_, node := range h.arr {

fmt.Printf("%d ", node)

}

}

func (h \*MinHeap) Parent(i int) int {

return (i - 1) / 2

}

func (h \*MinHeap) LeftChild(i int) int {

return 2\*i + 1

}

func (h \*MinHeap) RightChild(i int) int {

return 2\*i + 2

}

func main() {

h := &MinHeap{}

h.Insert(5)

h.Insert(3)

h.Insert(9)

h.Insert(6)

h.Insert(2)

h.Insert(1)

h.Insert(7)

h.Insert(8)

h.Insert(10)

h.Remove()

arr := []int{5,3,8,9,12,1,3,6,12,2}

h.Heapify(arr)

h.Display()

}